

## **Remarks**

In the Office Action, the Examiner noted that claims 1-39 are pending in the application, and that claims 1-39 are rejected. By this amendment, claims 14-18 have been canceled, and claims 1, 19, 27, and 39 have been amended. Thus, claims 1-13 and 19-39 are pending in the application.

Applicant hereby requests further examination and reconsideration of the application, in view of the foregoing amendments.

## ***In the Specification***

In the specification, the table on page 1 has been amended to identify the co-pending applications by their serial numbers.

## ***In the Claims***

### **Rejection Under 35 USC 112 second paragraph**

The Examiner rejected claim 18 under 35 U.S.C. § 112, second paragraph, as being indefinite. The Examiner's rejection is mooted by the cancellation of claim 18.

### **Rejection Under 35 USC 102(b)**

The Examiner rejected claims 1-7, 27, 33-35, and 38 under 35 U.S.C. § 102(b), as being anticipated by *Gochman, et al.*, U.S. Patent No. 5,969,868 (hereinafter *Gochman*). Applicant respectfully traverses the rejection of claims 1-13 and 19-39.

With respect to claim 1, the Examiner asserted that *Gochman* has taught a comparator that compares first and second return addresses generated by first and second call/return stacks. Applicant respectfully asserts that *Gochman* does not teach a comparator that compares first and second return addresses generated by first and second call/return stacks for the following reasons.

#### ***1. Gochman's Speculative Return Stack Buffer Does Not Provide the Final Return Address that is Compared With the Speculative Return Stack Buffer Predicted Return Address***

*Gochman* teaches a Speculative Return Address Stack that provides a predicted target address of a return instruction at the time the return instruction is fetched; the predicted

target address is used to speculatively fetch subsequent instructions. Col. 4, lines 34-44. Because the predicted target address may be wrong, the return instruction is subsequently executed by an execution stage of the processor that determines a final return address; the final return address is popped off “the microprocessor architectural stack.” Col. 7, lines 1-10. It is this final return address that the execution unit compares with the predicted target address. Col. 7, lines 20-22. The Examiner apparently asserts that the final return address (which the Examiner asserts corresponds to Applicant’s second return address of claim 1) is provided by *Gochman*’s Actual Return Stack Buffer. For the following reasons, Applicant respectfully asserts that “the microprocessor architectural stack” from which the final return address is popped does not refer to the Actual Return Stack Buffer; rather, the only structure taught in *Gochman* that may correspond to “the microprocessor architectural stack” is the architectural stack in main memory. Succinctly, *Gochman*’s microprocessor will not correctly execute certain programs if it uses the Actual Return Stack Buffer return address to detect mispredictions by the Speculative Return Stack Buffer because it is possible that *Gochman*’s Actual Return Stack Buffer may be inconsistent with the state of the architectural stack (i.e., main memory stack) due to certain irregular programming practices as explained below.

*Gochman* states that in most processors the stack to which call and return instructions push and pop return addresses is in a main memory coupled to the microprocessor, and the location of the stack in main memory is specified by a stack pointer register. Col. 2, lines 6-19. Because main memory accesses are slow, popping a return address from main memory causes a pipeline stall; therefore, some processors predict return addresses using a return stack buffer within the processor to reduce or eliminate the pipeline stalls. Col. 2, lines 21-37. *Gochman* goes on to describe how a return stack buffer may mispredict a return address, and in particular how a return stack buffer may become damaged, i.e., may not be consistent with the state of the stack in main memory. Col. 2, lines 38-56. Thus, the need for *Gochman*’s Actual Return Stack Buffer which, *Gochman* purports, contains architecturally correct information. Col. 4, lines 56-67. When the processor determines that the Speculative Return Stack Buffer has mispredicted a return address, the contents of the Actual Return Stack Buffer is copied to the Speculative Return Stack Buffer to “correct any corrupted information.” Col. 2, line 67 to col. 3, line 3.

*Gochman* teaches that the Actual Return Stack Buffer contains architecturally correct information because when his execution stage executes a call instruction it calculates a final return address that is pushed onto the Actual Return Stack Buffer, and when his execution stage executes a return instruction it pops the return address off the Actual Return Stack Buffer. However, certain irregular programming practices are well known that would cause *Gochman*'s Actual Return Stack Buffer to be inconsistent with the architectural main memory stack.

In particular, it is well-known that certain programs do not always execute call/return instructions in the regular fashion. In a regular program sequence, the calling routine executes a call instruction. The call instruction instructs the microprocessor to push a return address onto the main memory stack and then to branch to the address of the subroutine. The return address pushed onto the stack is the address of the instruction that follows the call instruction in the calling routine. The subroutine ultimately executes a return instruction, which pops the return address off the stack, which was previously pushed by the call instruction, and branches to the return address, which is the target address of the return instruction. Although there may be intervening stack accesses, even due to other call/return pairs, by the time the return instruction is executed, the stack pointer value is restored to its state just after the original call instruction was executed so that when the corresponding return instruction is executed the return address previously pushed onto the stack by the call instruction is popped off the stack and used as the current instruction pointer value.

One known program that does not execute call/return in the regular fashion includes a call instruction, then a push instruction to place a different return address on the stack, then a return instruction, which causes a return to the pushed return address rather than to the address of the instruction after the call instruction, which was pushed onto the stack by the call instruction. Another program performs a push instruction to place a return address on the stack, then performs a call instruction, then performs two return instructions, which causes a return to the pushed return address in the case of the second return instruction rather than to the instruction after a call instruction that preceded the push instruction. In these situations, because *Gochman* does not teach updating the Actual Return Stack Buffer with the value of the push instructions, the Actual Return

Stack Buffer would be inconsistent with the architectural main memory stack and would not contain at its top of stack the same value as the top of stack in main memory when the return instruction was executed. Consequently, if *Gochman* were to use the value at the top of the Actual Return Stack Buffer as the final return address for comparison with the predicted return address provided by the Speculative Return Stack Buffer, the possibility exists that the processor would not detect a misprediction; thus, the processor would not execute the program correctly.

In another example, a common programming practice on x86 processors (Applicant is not asserting *Gochman*'s processor is necessarily an x86 processor, but merely provides the example as an illustration of a irregular call/return sequence) is to execute a far call instruction, which pushes both a segment descriptor and a return address onto the stack. However, rather than executing a far return instruction (as would be typical), which would pop the segment descriptor and load it and then pop the return address, the subroutine executes a pop instruction (to pop the segment descriptor) followed by a near return, which pops the return address. The advantage is that if the programmer knows the program is remaining in the same privilege level (i.e., loading the descriptor would not change the privilege level), he avoids the overhead of loading a new descriptor. Again, in this situation because *Gochman* does not teach popping the Actual Return Stack Buffer in response to the pop instruction, the Actual Return Stack Buffer would be inconsistent with the architectural main memory stack and would not contain at its top of stack the same value as the top of stack in main memory when the return instruction was executed. Consequently, if *Gochman* were to use the value at the top of the Actual Return Stack Buffer as the final return address for comparison with the predicted return address provided by the Speculative Return Stack Buffer, the possibility exists that the processor would not detect a misprediction; thus, the processor would not execute the program correctly.

Thus, *Gochman* does not teach how to keep the Actual Return Stack Buffer from becoming inconsistent with the architectural stack in main memory when executing irregular programs such as the ones described above. Rather, *Gochman*'s invention is directed to solving a completely different problem, namely the side-effects of updating the Speculative Return Stack Buffer with information relating to call or return

instructions that are in the speculative execution path of branch instructions that are subsequently determined to have been mispredicted. By updating the Actual Return Stack Buffer only in response to actually executed instructions, rather than in response to speculatively executed instructions, the Actual Return Stack Buffer avoids the inconsistencies with the architectural main memory stack associated with speculative execution, but does not teach how to avoid the inconsistencies with the architectural main memory stack associated with program that execute call/returns in an irregular fashion, such as those described above.

In summary, if Applicant's assertion is true that *Gochman* does not teach the final return address being provided by the Actual Return Stack Buffer, then the Actual Return Stack Buffer does not provide a return address for comparison to anything; rather, the only use *Gochman* teaches of his Actual Return Stack Buffer return addresses are for copying to the Speculative Return Stack Buffer. Hence, *Gochman*'s Actual Return Stack Buffer return address is not compared with *Gochman* Speculative Return Stack Buffer return address; therefore, *Gochman* does not anticipate the limitation of claim 1 of comparing first and second return addresses provided by first *and second* return stacks. Rather, *Gochman* teaches comparing first and second return addresses provided by a return stack and by "the microprocessor architectural stack," and the main memory stack is the only other structure taught by *Gochman* that could be "the microprocessor architectural stack."

***2. Gochman Does Not Teach Comparing the First and Second Return Addresses Prior to Execution of the Return Instruction, as Recited by Amended Claim 1***

Even assuming *arguendo* that Applicant's assertion above is incorrect, i.e., assuming that *Gochman*'s Actual Return Stack Buffer provides the second return address, *Gochman* teaches that the comparison of the predicted return address and the final return address is performed in the execution stage of the processor. Col. 7, lines 20-22. Applicant has amended claim 1 to clarify that the first and second return addresses are compared prior to execution of the return instruction. That is, the amendment clarifies that the first and second return addresses are predictions since they are made prior to actual execution of the return instruction. In contrast, *Gochman*'s Actual Return Stack Buffer never actually makes a prediction of the return address. That is, *Gochman* does not teach the processor

branching to a return address provided by the Actual Return Stack Buffer; rather, the Actual Return Stack Buffer return addresses are simply used to update the Speculative Return Stack Buffer.

For the reasons stated above, Applicant respectfully asserts that *Gochman* does not anticipate claim 1.

Applicant respectfully asserts *Gochman* does not anticipate or obviate dependent claims 2-13 because they depend from independent claim 1, which is not anticipated or obviated by *Gochman* for the reasons discussed above.

Applicant respectfully asserts *Gochman* does not anticipate or obviate independent claim 19 for similar reasons discussed above with respect to claim 1, but more particularly that assuming *arguendo* the Actual Return Stack Buffer provides the final return address, it does not do so prior to execution of the return instruction.

Applicant respectfully asserts *Gochman* does not anticipate or obviate dependent claims 20-26 because they depend from independent claim 19, which is not anticipated or obviated by *Gochman* for the reasons discussed above.

Applicant respectfully asserts *Gochman* does not anticipate or obviate independent claims 27 or 39 for the same reasons discussed above with respect to claim 1.

Applicant respectfully asserts *Gochman* does not anticipate or obviate dependent claims 28-38 because they depend from independent claim 27, which is not anticipated or obviated by *Gochman* for the reasons discussed above.

### **Rejection Under 35 USC 103**

The Examiner rejected claims 9-13 and 36-37 under 35 U.S.C. § 103, as being unpatentable over *Gochman* in view of *Hilgendorf, et al.*, U.S. Patent No. 5,974,543 (hereinafter *Hilgendorf*). Applicant respectfully traverses the rejection of claims 9-13 and 36-37.

With respect to claim 11, the Examiner asserts that *Hilgendorf* has taught a BTAC that is configured to cache a plurality of byte offsets within an instruction cache line of a corresponding plurality of call instructions, the byte offsets being within an instruction cache line selected by an instruction cache fetch address, citing col. 5, lines 20-24.

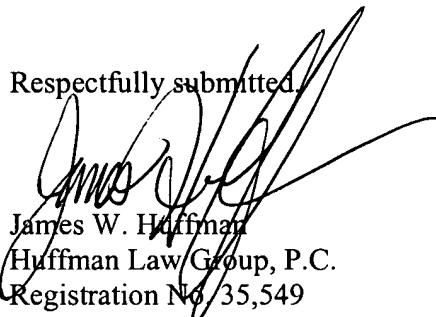
Applicant respectfully asserts that *Hilgendorf* does not teach a BTAC that caches a plurality of bytes offsets within an instruction cache line of a corresponding plurality of call instructions. The text cited by the Examiner teaches caching the *length* of call instructions, not the byte offset of the location of the call instruction within the cache line selected by the fetch address. Applicant can find no teaching in *Hilgendorf* of caching call instruction byte offsets. Therefore, Applicant respectfully asserts that *Gochman* in view of *Hilgendorf* does not obviate claim 11. The same is true with respect to claim 37.

The Examiner has indicated additional prior art which is made of record and not relied upon. None of these references anticipate or obviate applicant's invention.

Applicant respectfully requests that a timely Notice of Allowance be issued in this case.

Applicant earnestly requests the examiner to telephone him at the direct dial number printed below if the examiner has any questions or suggestions concerning the application.

Respectfully submitted,

  
James W. Huffman  
Huffman Law Group, P.C.  
Registration No. 35,549  
Customer No. 23669  
1832 N. Cascade Ave.  
Colorado Springs, CO 80907  
719.475.7103  
719.623.0141 fax  
jim@huffmanlaw.net

Date: 9-16-04

"EXPRESS MAIL" mailing label number ED 04135 949545. Date of Deposit  
9-20-04. I hereby certify that this paper is being deposited with the U.S.  
Postal Service Express Mail Post Office to Addressee Service under 37 C.F.R. §1.10 on  
the date shown above and is addressed to the U.S. Commissioner of Patents and  
Trademarks, Alexandria, VA, 22313.

By: Wicki J. Day